Computers and Mathematics with Applications 59 (2010) 3029–3038

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

An ILP formulation and genetic algorithm for the Maximum Degree-Bounded Connected Subgraph problem

Milena Bogdanović *

Teacher Training Faculty, University of Niš, Partizanska 14, 17500 Vranje, Serbia

ARTICLE INFO

Article history:
Received 30 April 2009
Received in revised form 13 February 2010
Accepted 16 February 2010

Keywords:
Connected subgraph
Combinatorial optimization
Integer linear programming
Genetic algorithm

ABSTRACT

A general instance of a Degree-Constrained Subgraph problem may be found in an edge-weighted or vertex-weighted graph G whereas the objective is to find an optimal weighted subgraph, subject to certain degree constraints on the vertices of the subgraph. This class of combinatorial problems has been extensively studied due to its numerous applications in network design. If the input graph is bipartite, these problems are equivalent to classical transportation and assignment problems in operational research. This paper is an illustration of a research of the *NP*-hard Maximum Degree-Bounded Connected Subgraph problem (MDBCS). It is a classical *NP*-complete problem. Moreover this paper offers a first integer linear programming formulation of the (MDBCS), and a formal proof that it is correct. A genetic algorithm for obtaining the optimal solution of (MDBCS) has also been provided. The proposed solution comprises a genetic algorithm (GA) that uses binary representation, fine-grained tournament selection, one-point crossover, simple mutation with frozen genes and caching technique.

1. Introduction

In this paper the *NP*-hard problem, as well as the Maximum Degree-Bounded Connected Subgraph (MDBCS), (listed in [1]), has been studied whereby an ILP model for solving this (MDBCS) problem is demonstrated.

In complexity theory, *NP* denotes the set of all (decision) problems solvable by a non-deterministic polynomial time algorithm. P denotes the set of all (decision) problems solvable by a deterministic polynomial time algorithm. *NP* problems are "hard" in the sense they are not currently solvable in deterministic polynomial time. It is an open question whether $NP = P$.

Throughout this paper the graph G is supposed to be simple, i.e. to be a finite undirected graph without loops or multiple edges.

To start with some *definitions* that will need further elaboration are provided.

The graph G is a pair (V, E), where V is a set of vertices and E a set of edges. If G is a directed graph (digraph) then each edge is an ordered pair $(u, v) \in E$ where $u, v \in V$. If G is an undirected graph then each edge is a two element set $\{u, v\} \in E$ where $u, v \in V$.

Or, if the graph G is a pair (V, E), where V is a set of vertices, and E is a set of edges between the vertices $E \subseteq \{(u, v)|u, v \in V\}$, then G is a directed graph. If a graph does not allow self-loops, adjacency is irreflexive, that is $E \subseteq \{(u, v)|u, v \in V \wedge u \neq v\}$.

Two vertices of a graph are adjacent if they are joined by an edge. Vertex v is adjacent to u if and only if $(u, v) \in E$. In an undirected graph where the edge is $\{u, v\}$, and hence $\{v, u\}$, v is adjacent to u and u is adjacent to v. A path between two

* Tel.: +381 17421313; fax: +381 17421633.
E-mail address: mb2001969@beotel.net.

0898-1221/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2010.02.021

vertices is a sequence of edges that begins at one vertex and ends at another vertex i.e. $v_1, v_2, \ldots, v_n$ is a path if $\{v_i, v_{i+1}\} \in E$ for $i = 1, \ldots, n-1$. A simple path passes through a vertex only once. A *cycle* is a path that begins and ends at the same vertex. A *simple cycle* is a cycle not passing through other vertices more than once.

A connected graph has a path between each pair of distinct vertices. A cycle in a directed graph is a path of length where at least 1 is such that $v_1 = v_n$. This cycle is *simple* if the path is simple. For undirected graphs, the edges must be distinct.

An undirected graph is connected if there is a path from every vertex to every other vertex. We can label the edges of a graph with numeric values, the graph is called a *weighted* graph.

A graph $G' = (V', E')$ is a subgraph of another graph $G = (V, E)$ iff $V' \subseteq V$, and $E' \subseteq E \wedge ((v_1, v_2) \in E' \rightarrow v_1, v_2 \in V')$. In general, a subgraph does not need to have all possible edges. If a subgraph has every possible edge, it is an *induced* subgraph.

An edge-induced subgraph is a subset of the edges of a graph G together with any vertices that are their endpoints. A vertex-induced subgraph (sometimes simply called an "induced subgraph") is a subset of vertices of a graph G together with any edges whose endpoints are both in this subset.

A *tree* is a special kind of undirected graph. That is, a tree is a connected undirected graph without cycles.

In graph theory, a tree is a connected acyclic graph; unless stated otherwise, trees and graphs are undirected. There is no one-to-one correspondence between such trees and trees as data structure. We can take an arbitrary undirected tree, arbitrarily pick one of its vertices as a root, make all its edges directed by making them point away from the root node – producing an arborescence – and assign an order to all the nodes. The result corresponds to a tree data structure. Picking a different root or different ordering produces a different one.

All trees are graphs, but not all graphs are trees.

A *spanning tree* of a connected undirected graph G is a subgraph of G that contains all of G's vertices and sufficiently of its edges to form a tree. There may be several spanning trees for a given graph. If we have a connected undirected graph with cycles, and we remove edges until there are no cycles, then we obtain a spanning tree.

In the mathematical field of graph theory, a spanning tree T of a connected, undirected graph G is a tree composed of all vertices and some (or perhaps all) of the edges of G. Informally, a spanning tree of G is a selection of edges of G that form a tree spanning every vertex. That is, every vertex lies in a tree, but no cycles (or loops) are formed. On the other hand, every bridge of G must belong to T. A spanning tree of a connected graph G can also be defined as a maximal set of edges of G that contains no cycle, or as a minimal set of edges that connect all vertices. In certain fields of graph theory it is often useful to find a minimum spanning tree of a weighted graph. Other optimization problems on spanning trees have also been studied, including the maximum spanning tree, the minimum tree that spans at least k vertices, the minimum spanning tree with at most k edges per vertex (Degree-Constrained Spanning Tree), the spanning tree with the largest number of leaves (closely related to the smallest connected dominating set), the spanning tree with the fewest leaves (closely related to the Hamiltonian path problem), the minimum diameter spanning tree, and the minimum dilation spanning tree.

A *spanning subgraph* of a graph $G = (V, E)$ is a subgraph with vertex set V. A spanning tree is a spanning subgraph that is a tree. Spanning trees have been found to be structures of paramount importance in both theoretical and practical problems [2].

A single graph can have many different spanning trees. We can also assign a weight to each edge, which is a number representing how unfavorable it is, and use this to assign a weight to a spanning tree by computing the sum of weights of edges in that spanning tree. As a result spanning trees of a connected graph have been the focus for extensive attention in graph theoretic research.

In graph theory, a *connected component* of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and to which no more vertices or edges can be added while preserving its connectivity. That is, it is a maximal connected subgraph. Any notation or definition not explained here can be found in [3].

Observations about graphs:

A connected undirected graph with n vertices must have at least $n-1$ edges. A connected undirected graph with n vertices and exactly $n-1$ edges cannot contain a cycle. A connected undirected graph that has n vertices and more than $n-1$ edges must contain a cycle.

We consider the undirected graphs $G = (V, E)$, where E is an edge set and V is the vertex set, and let $|V| = n$ and $|E| = m$. Let $G = (V, E)$ be a connected graph (although the graph does not have to be connected), with a weight function $w : E \rightarrow \mathbb{R}^+$. For simplicity, we simply say that G is a w-weighted graph. For any subset $E' \subseteq E$, we denote by G' the subgraph of G induced by E'; and we denote by $w(E')$ the sum of the weights of the edges in E', that is $w(E') = \sum_{e \in E'} w(e)$.

Observations about genetics algorithms:

Genetic algorithms (GA) are a family of algorithms using some genetic principles that are present in nature, in order to solve certain mathematical problems [4]. These natural principles are: inheritance, crossing, mutation, survival of the best custom (or survival of the fittest), migration, etc. These algorithms can be used for solving various classes of problems, because they are fairly general in nature. In this case, they are used in the optimization problem - finding the optimal parameters of a system. The genetic algorithm is applied to the final set of individuals called population. Each individual in the population is represented by a series of characters (genetic code) and corresponds to a solution in the search space. Encoding can be binary or of some other higher cardinality alphabet. Encoding solutions is an important step of genetic algorithm because an inadequate choice of code can lead to bad results regardless of the other structures of the algorithm. Genetic algorithms are therefore robust and adaptive methods in addition to other fields of application that can be used for solving combinatorial optimization problems. The central concept in the description of genetic algorithms is a population of

individuals, which is usually between 10 and 200. Each individual represents a possible solution in the search space of the problem (the space of all solutions). Each individual is presented as genetic code of a certain finite alphabet, with the most commonly used binary coding, where the genetic code consists of a series of bits.

Adaptation function (or fitness function) are assigned to each of the individuals and they are evaluating the quality of a given species, as well as individual solutions in the search space. The task of genetic algorithms is to provide a constant improving, from generation to generation, of absolute fitness of the whole population. It is realized by repeated application of genetic operators selection, cross and mutation, thus improving all the solutions of the particular problem.

2. Previous work

Beyond the aesthetic and theoretical appeal of Degree-Constrained Subgraph problems, the reason for such intensive study is rooted in their wide applicability in the areas of interconnection networks and routing algorithms, among others. For instance, given an interconnection network developed by an undirected graph, one may be interested in finding a small subset of nodes whit high degree of connectivity for each node.

The Maximum Degree-Bounded Connected Subgraph (MDBCS) takes as input a weight function $w : E \to \mathbb{R}^+$ and an integer $d \geq 2$. Let us assume for a subset $E' \subseteq E$ and $V' \subseteq V$ such that the subgraph $G' = (V', E')$ is connected, subgraph G' has a maximum degree of d, and $\sum_{e \subseteq E'} w(e)$ is maximized.

Example 1. Let $|V| = 20$ and $|E| = 10$ and $w : E \to \mathbb{R}^+$ is shown below:

$V = \{1, 2, 3, 4, 5, \ldots, 19, 20\}$ and

$E = \{\{1, 2\}, \{3, 4\}, \{5, 6\}, \{7, 8\}, \{9, 10\}, \{11, 12\}, \{13, 14\}, \{15, 16\}, \{17, 18\}, \{19, 20\}\}$.

Let E_w denote a set of edges with the value of weight function, between an unordered pairs $\{u, v\}$ of vertices. Therefore,

$E_w = \{(\{1, 2\}, 17), (\{3, 4\}, 15), (\{5, 6\}, 13), (\{7, 8\}, 11), (\{9, 10\}, 15), (\{11, 12\}, 21), (\{13, 14\}, 11), (\{15, 16\}, 20), (\{17, 18\}, 19), (\{19, 20\}, 14)\}$.

The optimal objective value in this example is 21.

The corresponding subgraph G' is represented in this way: $V' = \{11, 12\}$ and $E'_w = \{(\{11, 12\}, 21)\}$.

Example 2. Supposing function $w : E \to \mathbb{R}^+$ in the graph G is presented as follows: $V = \{1, 2, 3, 4, 5, \ldots, 14, 15\}$ and

$E_w = \{(\{1, 8\}, 23), (\{1, 9\}, 28), (\{2, 11\}, 7), (\{2, 15\}, 13), (\{2, 12\}, 21), (\{2, 4\}, 22), (\{3, 6\}, 10), (\{3, 11\}, 20), (\{3, 15\}, 24), (\{4, 12\}, 2), (\{4, 7\}, 9), (\{5, 13\}, 14), (\{5, 6\}, 25), (\{5, 9\}, 25), (\{7, 12\}, 7), (\{8, 9\}, 7), (\{10, 14\}, 12), (\{10, 11\}, 17), (\{10, 12\}, 43), (\{11, 15\}, 6)\}$.

The optimal objective value is 335.

The corresponding subgraph G' is done with:

$V' = \{2, 3, 4, 6, 7, 10, 11, 12, 14, 15\}$ and

$E'_w = \{(\{2, 11\}, 7), (\{2, 15\}, 13), (\{2, 12\}, 21), (\{2, 4\}, 22), (\{3, 6\}, 10), (\{3, 11\}, 20), (\{3, 15\}, 24), (\{4, 7\}, 9), (\{4, 12\}, 2), (\{10, 11\}, 17), (\{10, 12\}, 43), (\{11, 15\}, 6)\}$.

This problem is one the classical *NP*-hard problems listed in [1], but there have been no documented results so far, except for $d = 2$. For $d = 2$, the unweighted (MDBCS) problem corresponds to the Longest Path Problem. Indeed, given the input graph G (which can be assumed to be connected), let P and G' be optimal solutions of the Longest Path and ($MDBCS_2$) in G respectively.

Then observe that $|E(G')| = |E(P)|$ unless G is Hamiltonian, in which case $|E(G')| = |E(P)| + 1$.

Note that the Minimum Subgraph of Minimum Degree Problem (MSMD) is closely related to (MDBCS) Problem. Indeed, (MSMD) corresponds exactly to the dual (unweighted) node-minimization version of (MDBCS) [5].

Amidi et al. in [5], have proven that the (MDBCS) problem is not in the Apx for any $d \geq 2$. On the other hand, they offer an approximation algorithm for general unweighted graphs with the ratio $min\frac{m}{\log n}, \frac{nd}{2\log n}$ and an approximation algorithm for general weighted graphs with the ratio $min\frac{n}{2}, \frac{m}{d}$. The first algorithm uses an algorithm introduced in [6], and based on the *color-coding* method.

The class Apx consists of all *NP*-hard optimization problems that can be approximated within a constant factor. The subclass PTAS (Polynomial Time Approximation Scheme) contains the problems that can be approximated in polynomial time within the ratio $1 + \epsilon$ for any constant $\epsilon > 0$. Assuming $P \neq NP$, there is a strict inclusion of PTAS in Apx (for instance, the Vertex Cover is in $Apx \setminus PTAS$), hence an Apx-hardness result for a problem implies the non-existence of a PTAS.

Without the connectivity constraint, the (MDBCS) problem is known as to be in *P* using matching techniques [7].

The following lemma shows that the (MDBCS) problem is *NP*-complete.

Lemma 1 ([5]). *For all $d \geq 2$, $MDBCS_d$ restricted to the class of graphs for which any optimal solution contains at least 2 vertices of degree at most $d - 1$ is NP-complete.*

The paper [2] deals with the classical Degree-Bounded Spanning Tree problem. Rahman and Kaykobad in [2] define degree d bounded spanning tree as a spanning tree having no vertices of degree larger than d. The main results of that paper state that, if $G = (V, E)$ is a connected graph having the independence number d, where $d \geq 2$, and G has a degree-d-bounded spanning tree.

Matsuda and Matsumura in [8], give sufficient conditions for a graph to have degree-bounded trees. Let G be a connected graph and A a vertex subset of G. They denote by $\sigma_d(A)$ the minimum value of the degree sum in G of any k independent vertices in A by $w(G-A)$ the number of components in the induced subgraph $G-A$. The main result in this paper is Theorem 3, listed in the following section.

Win in [9] proved that for an integer $d \geq 2$, the connected graph G for which $\sigma_d(G) \geq |V(G)| - 1$ is valid, has a spanning d-tree.

Sau and Thilikos in [10] give subexponential parameterized algorithms on planar graphs for a family of problems that consist in, given a graph G, finding a connected (induced) subgraph H with bounded maximum degree, while maximizing the number of edges (or vertices) of H. These problems are natural generalizations of the Longest Path. Their approach uses bidimensionality theory combined with novel dynamic programming techniques over branch decompositions of the input graph. These techniques can be applied to a more general family of problems that deal with finding connected subgraphs under certain degree constraints.

3. New Integer Linear Programming (ILP) for the (MDBCS) problem

Let $G = (V, E)$ be an undirected graph, let $d \geq 2$ be an integer, and let $w : E \to \mathbb{R}^+$ be a weight function. The Maximum Degree-Bounded Connected Subgraph problem is in finding a subgraph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, that subgraph G' must be of degree at most d, such that the subgraph G' is connected and $\sum_{e \in E'} w(e)$ has a maximum value.

The constants in this ILP are: an array of the set E, and the values of the function w in them.

The variables for this (ILP), are:

$$x_e = \begin{cases} 1, & e \in E' \\ 0, & e \notin E', \end{cases}$$

$$y_e = \begin{cases} 1, & e \in T' \\ 0, & e \notin T', \end{cases}$$

where $e \in E$, and T' is the spanning tree for the subgraph G',

$$z_i = \begin{cases} 1, & i \in V' \\ 0, & i \notin V'. \end{cases}$$

An integer programming formulation of the Maximum Degree-Bounded Connected Subgraph problem is shown below. This is a new formulation.

The ILP model of the (MDBCS) can now be formulated as:

Determine

$$\max \sum_{e \in E} w(e) x_e \tag{1}$$

subject to:

$$\sum_{e \ni i} x_e \leq d, \quad \forall i \in V, \tag{2}$$

$$y_e \leq x_e, \quad \forall e \in E, \tag{3}$$

$$\sum_{e \in E} y_e = -1 + \sum_{i \in V} z_i, \tag{4}$$

$$x_e \leq z_{i_e}, \quad \forall e \in E, \tag{5}$$

$$x_e \leq z_{j_e}, \quad \forall e \in E, \tag{6}$$

$$\sum_{i_e, j_e \in S} y_e \leq |S| - 1, \quad \forall S \subseteq V, |S| \geq 3. \tag{7}$$

The graph G is not oriented, and all vertices are doubles ones between the node. For this (ILP), in (5), i_e denotes the starting node of the edge e and in (6), j_e denotes the incoming node of the edge e.

The objective function (1) maximizes the sum of overall weights.

Recall that the undirected graph $G = (V, E)$ consists of two sets, a set V of vertices (nodes) and a set E of edges, that is, unordered pairs u, v or simply uv of vertices.

The condition in (2), guarantees that in the subgraph G' every node contains mostly k-vertices that lead from G'.

The condition in (3), guarantees that the subgraph G' is the preset of T'. Now, if T' is a spanning tree, and if it is connected, it follows that the subgraph G' is connected too.

In (4)–(6), we ensure that the spanning tree T' has as much vertices as the subgraph G' has edges minus 1.

The condition (7), guarantees that the spanning tree T' does not contain any cycle.

At last, (4)–(7), jointly provide that T' is the spanning tree for G'.

Lemma 2 ([8]). *Let $d \geq 2$ be an integer, G a connected graph, and $A \subseteq V(G)$. If $\sigma_d(A) \geq |V(G)| - 1$, then G has a tree T with maximum degree at most d and $A \subseteq V(T)$.*

In the previous Lemma 2, $V(G)$ denote a vertex set of graph G, A denote a vertex subset of G, and $\sigma_d(A)$ denote the minimum value of the degree sum in G of any d independent vertices in A. This lemma is a generalization of the result by Win [9].

Theorem 3. *The Maximum Degree-Bounded Connected Subgraph problem can be resolved if and only if constraints (2)–(7) are satisfied.*

Proof. ($\Rightarrow$): Let $G' = (V', E')$ be a d-bounded connected subgraph of G and $T' = (V', E(T'))$ be a spanning tree of G'. If values x_e, y_e, z_i are defined such as the above mentioned, then these values obviously satisfy constraints (2)–(7).

($\Leftarrow$): Further, let values x_e, y_e, z_i represent a feasible solution of ILP model, i.e. these values satisfy (2)–(7), and let $E' = \{e \in E : x_e = 1\}$ and $V' = \{i \in V : \exists e \in E' \wedge i \in e\}$. Then, it should be proven that $G' = (V', E')$ is a connected d-bounded subgraph of G. □

(i) It follows directly from (2) that each node in G' has a degree not greater than d;

(ii) If $E(T') = \{e \in E : y_e = 1\}$ and $V(T') = \{i \in V : \exists e \in E(T') \wedge i \in e\}$, than, according to (3), $T' = (V(T'), E(T'))$ is a subgraph of G';

(iii) It follows from (5) and (6) that

$$|V(T')| \leq |V| \leq \sum_{i \in V} (z_i); \tag{8}$$

(iv) From (7) for $S = V(T')$ and (8) it follows

$$|E(T')| \leq |V(T')| - 1 \leq \sum_{i \in V} (z_i) - 1; \tag{9}$$

(v) From (4) and (9) we have

$$|V(T')| = \sum_{i \in V} (z_i), \tag{10}$$

and, consequently,

$$|E(T')| = |V(T')| - 1; \tag{11}$$

(vi) From (11) and (7) it follows that T' is a tree.

(vii) From (8) and (10) we have that $|V(T')| = |V(T)|$, i.e. T' is a spanning tree of G' and, consequently, G' is connected.

The correctness of the ILP model for the (MDBCS) problem can be considered in the following way: Suppose that the (MDBCS) problem is resolved. It means that for any undirected graph $G = (V, E)$ we can find a corresponding subgraph $G' = (V', E')$, whit $V' \subseteq V$ and $G' \subseteq G$. That subgraph G' is connected and $\sum_{e \subseteq E'} w(e)$ is maximized. Therefore, when G' is connected, then T' like its spanning tree must be connected too, and it must hold $G' \supseteq T'$. It means that $y_e \leq x_e$ holds for each $e \in E$, i.e. constraint (3) is satisfied.

Furthermore, this means that in subgraph G', every node takes its most k vertices from it. Therefore, on the basis of Lemma 2, [8], the sum of all vertices from any node in subgraph G' cannot be other than k. In other words, $\sum_{e \ni i} x_e \leq k$ for each $i \in V$, i.e. constraint (2) is satisfied.

Since T' is the spanning tree for the subgraph G', then there are as much vertices in T' as edges in the subgraph G' minus 1. It means that constraint (4) is satisfied. Furthermore, for each edge $e = (i_e, j_e)$ is holding $x_e \leq z_{i_e}$ and $x_e \leq z_{j_e}$, for each $e \in E$. Therefore, if $E(T') = \{e \in E : y_e = 1\}$ and $V(T') = \{i \in V : \exists e \in E(T') \wedge i \in e\}$, than, according to (3), $T' = (V(T'), E(T'))$ is a subgraph of G'. Thus, constraints (5) and (6) are satisfied.

If T' is a tree, than there isn't a cycle, and all its subgraphs are, either simple trees or set of trees (and no cycles), so the number of edges is less or equal than the number of vertices minus 1.

Assume now that constraints (2)–(7) are satisfied and that there is evidence that the (MDBCS) problem is solved.

If the following sets are defined: $E' = \{e \in E | x_e = 1\}$, $T' = \{e \in E | y_e = 1\}$, $V' = \{i \in V | z_i = 1\}$, from the definition of variables it may be concluded that $\sum_{e \in E'} w(e)$, and $\max \sum_{e \in E} w_e x_e$ are equivalent.

If constraint (2) holds, then we obtain that in the subgraph G' every node has the most k vertices from it, by means of which we may guarantee solidity of the condition from the (MDBCS) problem. When constraint (3) holds, we may guarantee that

$G' \supseteq T'$. Constraint (4)–(7) guarantees that T' is the spanning tree. So, constraint (3)–(7) guarantees that T' is the spanning tree for G'. Forasmuch as T' is connected like the spanning tree, then the subgraph G' is connected too. Moreover, we obtain the subgraph which should be the result of the (MDBCS) problem. Further, from constraints (4)–(6) we obtain that graph T' has as much vertices as the subgraph G' has edges minus 1. It is recommendable that T' does not have any cycle, and if constraint (7) holds, this is correct. It follows that T' is the tree and the spanning tree for G' as though T' contains all nodes from G' because the spanning tree is a subgraph with vertex set V'.

It should be noted that all constraints (7) do not need to be inserted in this ILP, but only those which make a sense. For example, the number of variables on the left side are greater or equal to values on the right side, or if it is evidentially true, it removes them all.

4. The genetic algorithm for solving MDBCS problem

Since the (MDBCS) is an *NP*-hard problem, the ILP formulation (1)–(7) can be solved to optimality only for smaller dimension problem instances. In order to handle large-scale MDBCS instances, a heuristic approach has to be applied. We will use here a genetic algorithm (GA) methodology, which is one of the most popular metaheuristic approaches.

Genetic algorithms are stochastic search techniques that imitate some spontaneous optimization processes in the natural evolution. At each iteration, GA works with a set of individuals, named population. Each individual in the population represents an encoded solution of a problem. The initial population is either randomly or heuristically generated. The individuals' quality in the current population is evaluated by using a fitness function. Good individuals are selected to produce a new generation, by applying genetic operators crossover and mutation. The new individuals — offspring replace some of the individuals from the current population. The described process is iteratively preformed until some stopping criterion is satisfied. Detailed description of genetic algorithms can be found in [11]. Extensive computational experience on various optimization problems shows that GA often produces high quality solutions in a reasonable time [12–18]. Moreover, GA has shown to be robust with respect to parameter choice in reasonable bounds on quite different problems [19–21,13, 22,23,18].

The input for the algorithm is the undirected graph $G = (V, E)$, the function of weight (weight function) $w : E \to \mathbb{R}^+$ and an integer $d \geq 2$. Each edge of the graph is encrypt a zero (0) or unit (1) (use therefore, a binary encoding of individuals). If you find a node that has more than d edges, then it does not count, i.e. set the value 0, in order to get connecting components. Let's apply the search on the depth (or width). If the graph has only one component connection, then this is the end, i.e. the graph is connected and let's calculate the sum of all the weight of its edges and components. But if the relationship has more components, then take them in a row, the first and second, second and third, etc. and require the largest edge of the weight that connects the two components. Then take all edges of the shortest path from the original graph, and count only those with degree $\leq d$. The process continues until they connect all the components connection. At the end, all gather and get $\sum_{e \in E'} w(e)$.

The genetic operators used here are: fine-grained tournament selection [20], a one-point crossover, a simple mutation of the frozen genes and cache techniques [24,25]. Genetic algorithm is coded in the programming language C.

The outline of our GA implementation is given below, where N_{pop} denotes the overall number of individuals in the population, N_{elite} is a number of elite individuals and *ind* and obj_{ind} are the individual and its objective value.

```
Input_Data();
Population_Init();
while not Stopping_Criterion() do
    for ind := (N_elite + 1) to N_pop do
        if (Exist_in_Cache(ind)) then
            obj_ind := Get_Value_From_Cache(ind);
        else
            obj_ind := Objective_Function(ind);
            Put_Into_the_Cache_Memory(ind,obj_ind);
            if (Full_Cache_Memory()) then
                Remove_LRU_Block_From_Cache_Memory();
            endif
        endif
    endfor
    Fitness_Function();
    Selection();
    Crossover();
    Mutation();
endwhile
Output_Data();
```

The selection operator chooses the individuals that will produce offspring in the next generation, according to their fitness. Low fitness-valued individuals have less chance to be selected than high fitness-valued ones. In the standard

tournament scheme, one tournament is performed for every non-elitist individual. The tournament size is a given parameter and tournament candidates are randomly chosen from the current population. Only the winner of the tournament, i.e. a tournament candidate with the best fitness, participates in the crossover. The tournament is performed N_{nnel} times on the set of all N_{pop} individuals in the population to choose the N_{nnel} parents for crossover. The same individual from the current generation may participate in several tournaments. The standard tournament selection uses an integer tournament size, which in some cases may be a limiting factor.

Usually GAs have a relatively small number of elite individuals, because they have a chance to pass into the next generation twice: once through selection operator and once as elite individuals. Such common practice is not adequate for this purpose. In order to obtain satisfactory results of the GA implementation, it is necessary to provide a sufficient number of elite individuals to preserve good solutions for exploitation as well as sufficient number of non-elite individuals for exploration. To prevent an undeserved domination of N_{elite} elite individuals over the population, their fitness is decreased by the next formula:

$$f_{ind} = \begin{cases} f_{ind} - \bar{f}, & f_{ind} > \bar{f} \\ 0, & f_{ind} \leq \bar{f} \end{cases}; \quad 1 \leq ind \leq N_{elite};\ \bar{f} = \frac{1}{N_{pop}} \sum_{ind=1}^{N_{pop}} f_{ind}. \tag{12}$$

In this way, even non-elite individuals preserve their chance to pass to the next generation. The described approach allows high elitism without too high selection pressure, which may lead to over-exploitation in the algorithm.

The elitist strategy is applied to N_{elite} elite individuals, which are directly passing to the next generation. The genetic operators are applied to the rest of the population ($N_{nnel}N_{pop} - N_{elite}$ non-elite individuals). The objective value of elite individuals are the same as in the previous generation, so they are calculated only once, providing significant time-savings.

Duplicated individuals, i.e. individuals with the same genetic code are redundant. In order to prevent them to enter the next generation their fitness values are set to zero, except for the first occurrence. Individuals with the same objective value, but different genetic codes, in some cases may dominate the population by number, which implies that other individuals with potentially good genes are rare. For that reason, it is useful to limit the number of their appearance to some constant N_{rv}. This is a very effective technique for saving the diversity of the genetic material and keeping the algorithm away from a premature convergence. It consists of two steps for every individual in the population:

Step 1: Check whether the genetic code of the current individual *ind* is identical with the genetic code of any of the individuals from 1 to $ind - 1$. If the answer is positive, set the fitness of *ind* to 0. Otherwise go to *Step 2*;

Step 2: Count the number of the individuals from 1 to $ind - 1$ which did not get fitness 0 in *Step 1* and which have the same objective value as *ind*. If it is greater than or equal to N_{rv}, set the fitness of *ind* to 0.

The selection operator chooses individuals that will produce offspring in the next generation, according to their fitness. Low fitness-valued individuals have less chance to be selected than high fitness-valued ones. In the standard tournament scheme, one tournament is performed for every non-elitist individual. The tournament size is a given parameter and tournament candidates are randomly chosen from the current population. Only the winner of the tournament, i.e. a tournament candidate with the best fitness, participates in the crossover. The tournament is performed N_{nnel} times on the set of all N_{pop} individuals in the population to choose the N_{nnel} parents for crossover. The same individual from the current generation may participate in several tournaments. The standard tournament selection uses an integer tournament size, which in some cases may be a limiting factor.

The improved tournament selection operator, also known as the fine-grained tournament selection — FGTS [20], is implemented in Selection(). This operator uses a real (rational) parameter F_{tour} which denotes the desired average tournament size. The first type of tournaments is held k_1 times and its size is $\lfloor F_{tour} \rfloor$, while the second type is performed k_2 times with $\lceil F_{tour} \rceil$ individuals participating, so $F_{tour} \approx \frac{k_1 \cdot \lfloor F_{tour} \rfloor + k_2 \cdot \lceil F_{tour} \rceil}{N_{nnel}}$.

Extensive numerical experiments in [20,21,12,18] performed for different optimization problems, indicate that FGTS gives the best results for $F_{tour} = 5.4$. For that reason, the same value is used as reasonable choice in this GA implementation. The running time for the FGTS operator is $O(N_{nnel} \cdot F_{tour})$. In practice F_{tour} and N_{nnel} are considered to be constant (not depending on n) that gives a constant running time complexity. For detailed information about FGTS see [21].

In Crossover() all non-elitist individuals chosen to produce offspring for the next generation are randomly paired for exchanging genes in $\lfloor N_{nnel}/2 \rfloor$ pairs. After a pair of parents is selected, a crossover operator is applied to them producing two offsprings. The standard one-point crossover operator is used in this GA implementation. This operator is performed by exchanging segments of two parents' genetic codes starting from a randomly chosen crossover point. The crossover operator is realized with the probability $p_{cross} = 0.85$. It means that approximately 85% pairs of individuals exchange their genetic material.

The standard simple mutation operator is implemented in the proposed GA. It is performed by changing a randomly selected gene in the genetic code of the individual, with a certain mutation rate. During the GA execution it may happen that all individuals in the population have the same gene on a certain position. This gene is called frozen. If the number of frozen genes is l, the search space becomes 2^l times smaller and the possibility of a premature convergence rapidly increases. The crossover operator can not change the bit value of any frozen gene and the basic mutation rate is often too small to restore lost subregions of the search space. On the other hand, if the basic mutation rate is increased significantly, a genetic algorithm becomes a random search.

Table 4.1

d	Running time in seconds	Value of objective function
2	0.000000	21.000000
3	0.000000	21.000000
4	0.000000	21.000000

Table 4.2

d	Running time in seconds	Value of objective function
2	1.468000	253.000000
3	1.500000	326.000000
4	0.000000	335.000000
5	0.000000	335.000000
6	0.000000	335.000000

For that reason, the simple mutation operator used in Mutation() is modified so that the mutation rate is increased only on frozen genes. In this implementation, the mutation rate for frozen genes is increased 2.5 times $(1.0/n)$, comparing to non-frozen ones $(0.4/n)$. In each generation, we determine positions where all individuals have a given gene fixed and define them as frozen genes. Obviously the set of frozen genes is not fixed, i.e. it may change in a generation.

The initial population is randomly generated in Population_Init(). This approach provides maximal diversity of the genetic material. This function also computes values of all the individuals of the population.

In order to obtain satisfactory results of our GA implementation, it is necessary to have a sufficient number of elite individuals to preserve good solutions for exploitation, as well as sufficient number of non-elite individuals for the exploration. The population size of $N_{pop} = 150$ individuals with $N_{elite} = 100$ elite and $N_{nnel} = 50$ non-elite individuals is a good compromise between the exploitation and exploration part of the GA search. In this case, the corresponding values of k_1 and k_2 in FGTS operator are 20 and 30, respectively. The maximal allowed number of individuals with the same objective value is $N_{rv} = 40$.

During the running of GA, some solutions are often generated repeatedly in subsequent generations. The classical GA evaluates each solution irrespectively of its repetition. To increase the efficiency, we used a caching technique. The main idea is to avoid computing the same objective value every time when genetic operators produce individuals with the same genetic code. Evaluated objective values are stored in a hash-queue data structure using the least recently used (LRU) caching technique. When the same code is obtained again, its objective value is taken from the cache memory, that provides time-savings. In this implementation the number of individuals stored in the cache memory is limited to 5000. Function Exist_in_Cache(*ind*) investigates whether the cache memory contains individual *ind*. In that case the objective value obj_{ind} is directly taken from the cache memory by Get_Value_From_Cache(*ind*). Otherwise, the objective value obj_{ind} is calculated and the pair (ind, obj_{ind}) is stored in the cache memory by Put_Into_the_Cache_Memory(ind, obj_{ind}). If the cache memory is full, in order to make space for a new entry, function Remove_LRU_Block_From_Cache_Memory() removes the block (ind, obj_{ind}), which has been the least recently used. For detailed information about caching GA, see [26].

We used the program for the software package CPLEX for checking the results of the mathematical models (1)–(7). The genetic algorithm is tested on the test-examples reached the same value as the CPLEX program, and as the running time was very short in both cases, information on running times are presented.

Testing genetic algorithm for solving the MDBCS problem for Examples 1 and 2 in the section *Previous works*, provided the following results shown in Tables 4.1, 4.2, 4.3a and 4.3b.

For the graph from Example 1, the following results are obtained using a genetic algorithm for the MDBCS problem:

For the graph from Example 2, we have the following results:

The data in columns of Table 4.3a and of Table 4.3b represent, respectively:

- the name of the current instances containing dimension;
- the cardinality of the set of vertices, n;
- the cardinality of a set of edges, m;
- an integer $d \geq 2$, d;
- the best GA solution, GA_{best};
- the average time t (in seconds) needed to detect the best value;
- the total time t_{tot} (in seconds) needed to finish the GA;
- the total number of generations, gen;
- the average percentage of cache using, *cache*.

Since there are no standard instances for MDBCS, the instances for the k-cardinality tree problem, (KTCP) instances containing vertex-weights appropriate for our problem with necessary updates in accordance with the input for GA, have been used.

The stopping criterion of the GA was the maximum number of generations equal to 5000 or at most 2000 generations without improvement of the objective value.

Table 4.3a

Instance name	n	m	d	GA_{best}	t (s)	t_{tot} (s)	gen	$cache$ (%)
proba10-13	10	13	5	75	0.00	0.48	2004	98.189337
proba12-17	12	17	4	128	0.00	0.52	2011	96.627607
g15-4-01	15	20	4	335	0.01	0.52	2011	95.627607
g25-4-01	25	5	4	984	0.03	0.56	2050	82.505602
liter28-31	28	31	4	31	0.00	0.53	2028	88.879370
g30-4-01	30	229	22	11 911	0.25	1.98	2213	48.078520
liter34-39	34	39	6	371	0.01	0.53	2033	84.185658
w40-40	40	40	9	1 120	0.00	0.52	2026	85.130606
proba40-40	40	40	6	328	0.01	0.64	2024	57.248150
h44-44	44	44	7	377	0.01	0.55	2038	85.138658
w50-50	50	50	9	1 359	0.01	0.58	2040	80.374939
w60-60	60	60	9	1 651	0.01	0.61	2038	77.431651
w65-65	65	65	9	1 801	0.03	0.72	2077	75.603846
h69-69	69	69	7	771	0.03	0.72	2064	74.507983
w70-70	70	70	9	1 933	0.03	0.73	2081	74.129559
w85-85	85	85	9	2 252	0.06	0.89	2116	67.578103

Table 4.3b

Instance name	n	m	d	GA_{best}	t (s)	t_{tot} (s)	gen	$cache$ (%)
w100-100	100	100	9	2591	0.08	1.03	2118	64.506365
w115-115	115	115	9	2591	0.08	1.27	2102	50.347743
h118-118	118	118	8	2091	0.08	1.16	2117	61.187736
w130-130	130	130	9	2591	0.09	1.41	2117	46.924528
h146-146	146	146	9	3097	0.17	1.48	2183	56.603843
w150-150	150	150	9	2591	0.13	1.55	2140	44.107326
ha150-150	50	50	7	3704	0.19	1.55	2185	54.938757
w155-155	155	155	9	2591	0.11	1.56	2125	42.615602
wax50-1-168-168	168	168	9	2591	0.09	1.67	2108	39.947892
h183-183	183	183	10	4457	0.39	2.02	2367	51.837131
h196-196	196	196	10	5027	0.33	2.14	2286	50.710354
h201-201	201	201	10	5260	0.20	2.05	2169	50.557090
h212-212	212	212	10	5686	0.36	2.25	2291	48.965126
h44-48	44	48	8	489	0.01	0.55	2032	82.671253
h148-152	148	152	7	3941	0.19	1.56	2191	54.732908
w70-76	70	76	9	2224	0.03	0.75	2064	72.957910

5. Conclusions

This paper deals with one of the known Degree-Constrained Subgraph problems, the Maximum Degree-Bounded Connected Subgraph problem. The paper also describes the ILP model and the genetic algorithm for solving the MDBCS problem.

Here, a first integer linear programming formulation for this problem, is presented, with formal proof of its correctness. In this paper a GA approach to solving the MDBCS has also been described. The problem is solved by a genetic algorithm which includes a binary representation with frozen genes, a limited number of different individuals with the same objective value and the caching GA technique. Experimental results are very encouraging and show the effectiveness of the GA approach.

This research can be extended in several ways. It would be desirable to investigate the exact methods based on linear relaxation, as Branch-and-Cut, or Brunch-and-Cut-and-Price.

Acknowledgement

The author is grateful to Jozef Kratica for his useful suggestions and comments and general help in preparing this paper.

References

[1] M. Garey, D. Johnson, Computers and Intractability, W.H. Freeman, San Francisco, 1979.
[2] M.S. Rahman, M. Kaykobad, Independence number and degree bounded spanning tree, Applied Mathematics E-Notes 4 (2004) 122–124.
[3] D.B. West, Introduction to Graph Theory, Prentice Hall, Inc., New Jersey, 1996.
[4] J.H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, 1975.
[5] O Amidi, D. Peleg, S. Pérennes, J San, S. Saurabh, Degree-Constrained Subgraph problems hardness and approximation results, 2008, 1–29. http://hal.inria.fr/inria-00331747/en/.
[6] N. Alon, R. Yuster, U. Zwick, Color-coding: A new method for finding simple paths, cycles and other small subgraphs within large graphs, in: Proceedings of the 26th Annual ACM Symposium on Theory of Computing, STOC, 1994, pp. 326–335.
[7] L. Lovász, M. Plummer, Matching theory, in: Annals of Discret Mathematics, North-Holland, Amsterdam, 1986.

[8] H. Matsuda, H. Matsumura, Degree conditions and degree bounded trees, Discrete Mathematics, (2008), 295–298. doi:10.1016/j.disc.2007.12.099.
[9] S. Win, Existenz von Gerüsten mit Vorgeschriebenem Maximalgrad in Graphen, Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg 43 (1975) 263–267.
[10] I. Sau, D.M. Thilikos, Subexponential parameterized algorithms for bounded-degree connected subgraph problems on planar graphs, Electronic Notes in Discrete Mathematics 32 (2009) 59–66.
[11] M. Mitchell, Introduction to Genetic Algorithms, MIT Press, Cambridge, Massachusetts, 1999.
[12] V. Filipović, Operatori selekcije i migracije i Web servisi kod paralelnih evolutivnih algoritama, Ph.D. Thesis, University of Belgrade, Faculty of Mathematics, 2006.
[13] J. Kratica, Z. Stanimirović, D. Tošić, V. Filipović, Two genetic algorithms for solving the uncapacitated single allocation p-hub median problem, European Journal of Operational Research 182 (2007) 15–28.
[14] I. Ljubić, G.R. Raidl, A memetic algorithm for minimum-cost vertex-biconnectivity augmentation of graphs, Journal of Heuristics 9 (2003) 401–427.
[15] I. Ljubić, Exact and memetic algorithms for two network design problems, Ph.D. Thesis, Institute of Computer Graphics, Vienna University of Technology, 2004.
[16] J. Puchinger, G.R. Raidl, U. Pferschy, The core concept for the multidimensional knapsack problem, in: Lecture Notes in Computer Science, vol. 3906, 2006, pp. 195–208.
[17] G.R. Raidl, J. Gottlieb, Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem, Evolutionary Computation 13 (2006) 441–475.
[18] Z. Stanimirović, J. Kratica, Dj. Dugošija, Genetic algorithms for solving the discrete ordered median problem, European Journal of Operational Research 182 (2007) 983–1001.
[19] B. Djurić, J. Kratica, D. Tošić, V. Filipović, Solving the maximally balanced connected partition problem in graphs by using genetic algorithm, Computing and Informatics 27 (2008) 341–354.
[20] V. Filipović, J. Kratica, D. Tošić, I. Ljubić, Fine grained tournament selection for the simple plant location problem, in: Proceedings on the 5th Online World Conference on Soft Computing Methods in Industrial Application — WSC5, 2000, pp. 152–158.
[21] V. Filipović, Fine-grained tournament selection operator in genetic algorithms, Computing and Informatics 22 (2003) 143–161.
[22] J. Kratica, V. Kovačević-Vujčić, M. Čangalović, Computing the metric dimension of graphs by genetic algorithms, Computational Optimization and Applications 44 (2009) 343–361. doi:10.1007/s10589-007-9154-5.
[23] J. Kratica, V. Kovačević-Vujčić, M. Čangalović, Computing strong metric dimension of some special classes of graphs by genetic algorithms, Yugoslav Journal of Operations Research 18 (2008) 143–151.
[24] J. Kratica, Improvement of simple genetic algorithm for solving the uncapacitated warehouse location problem, in: R. Roy, T. Furuhachi, P.K. Chawdhry (Eds.), Advances in Soft Computing — Engineering Design and Manufactining, Springer-Verlang London Limited, 1999, pp. 390–402.
[25] J. Kratica, Parallelisation genetic algorithms for solving some *NP*-complete problems, Ph.D. Thesis, Faculty of Mathematics, Belgrade, 2000.
[26] J. Kratica, Improving performances of the genetic algorithm by caching, Computers and Artificial Intelligence 18 (1999) 271–283.